# Criticality Aware Access Control Model for Pervasive Applications *

S. K. S. Gupta, T. Mukherjee and K. Venkatasubramanian
Dept. of Computer Science and Engineering
Arizona State University
Tempe, AZ 85287
(sandeep.gupta)@asu.edu

## Abstract

*In this paper we present a new framework for specifying access control policies in smart spaces called Criticality-Aware Access Control (CAAC). The main idea is to automatically respond to occurrences of critical events within the smart space and change the access control policies accordingly. Current solutions like Context Aware-Role Based Access Control(CA-RBAC) are not designed to take critical events into account. CAAC extends the CA-RBAC model by including a new parameter called* Criticality *which measures the urgency of tackling the effects of a critical event.*

*We further identify five basic requirements for handling critical events:* Responsiveness*,* Correctness*,* Non-interference*,* Liveness *and* Non-repudiability*. Based on the CAAC framework, we define a sample set of access control policies and validate them to show that they meet the aforementioned requirements.*

## 1 Introduction

An important aspect of Ubiquitous Computing is to develop intelligent environments which allow inhabitants to interact seamlessly with a smart, information-rich space. An example of a smart space can be a 'smart hospital emergency department (ED)', where patients are automatically and continuously monitored, and any emergency is immediately reported to medical professionals. The development of smart spaces raises many security issues. The ability of such spaces to monitor and interact with its internal elements, makes them ideal candidates for hackers and tech-criminals to exploit. Access control is therefore an essential component of smart spaces to prevent unauthorized access to the information available in them.

Any system, including smart spaces, is in one of at least two states: normal or abnormal. In a normal state, the system provides services in response to routine events. A smart-ED, responding to the arrival of new critical patients, may automatically allocate appropriate resources, such as available beds, contact emergency personnel, display the location of necessary equipment, and so on. Access control in such normal states can follow standard pre-defined policies, such as Role-Based Access Control [3]. These models can be effective in providing access control during such normal system states.

However, when the system is in an abnormal state, its service requirements may change radically. Routine services provided by the system may not meet the demands of the new conditions. Allocating resources when a natural disaster brings an influx of a large number of patients to the smart-ED is a far more demanding task. Standard access control policies may prevent medical personnel from using the resources they need to respond to the emergency. For example, nurses brought in from neighboring hospitals in response to a disaster may not automatically be granted access to equipment in the smart-ED.

A smart space needs to be able to observe its environment continually and to take corrective measures in case of environmental changes which can cause the system to enter an abnormal state. We call such environmental changes *Critical Events*. Once a critical event has been detected, the system should take immediate (as soon as the criticality is observed) action to control its effects. In the case of fire, the smart-ED may not only notify the fire-department, but also perform other essential functions, such as prioritize the

*Supported in part by MediServe Information Systems, Consortium for Embedded Systems and National Science Foundation

evacuation of patients and associated equipment. Critical events have a finite period during which corrective action must be taken. For example during a fire in the smart-ED, patients need to be evacuated ASAP. The system response to critical events should not interfere with the normal working of the system. In the case of a large patient influx due to a disaster, the treatment of patients already in the ED should not suffer. Further, any system changes in response to a critical event should be temporary. Any access control policies modified in smart-ED during an emergency should be restored once the emergency is over.

One of the first efforts in providing access control in a smart space was the Role Based Access Control model introduced in [3]. In RBAC, subjects in the system are assigned *roles* when they join the system, and are allowed to access resources, within the system, based on the *privileges* associated with the roles. Given a set of roles and privileges RBAC involves two main activities: mapping subjects to roles and mapping the roles to different sets of privileges. In traditional RBAC, the two mappings are static and do not reflect dynamic changes within the system.

Context-aware Role Based Access Control (CA-RBAC) [5] was designed to make RBAC dynamic by including context information while associating subjects to roles and roles to privileges. To provide these context aware mappings, traditional CA-RBAC include context information (like space, time, resource capability, subject capability etc.) while making access control decisions. However, in CA-RBAC context information is gathered only as a response to explicit access requests within the system. This model does not react to the occurrence of critical events, which, as we have seen, have unique access control requirements.

To meet the demands of critical events produced from abnormal system states, a new access control model for smart spaces is needed. This model should not only prevent unauthorized access within the space, but also provide facilities for responding to the effects of critical events. *The goal of this paper is therefore to develop a generic framework for specifying access control policies that perform the dual task of controlling access to resources during both critical and non-critical events within the smart space.*

### 1.1 Contribution and Organization of the Paper

Our contribution is the development of a new access control model for smart spaces which extends the traditional CA-RBAC model to take into account the effects of critical events. We call this new model *Criticality Aware Access Control (CAAC)*. We introduce a new term, *Criticality*, that defines the level of urgency required for various mitigative actions during a critical event[1]. CAAC classifies system context information into two main categories: critical and non-critical. Critical contexts indicate the setting or occurrence of a critical event which requires immediate action. Non-critical contexts, on the other hand, are observed during normal system states and require no special action (In the sequel, we use the terms critical event, and critical context interchangeably). CAAC provides the ability to automatically adjust access control policies in response to critical events. This may include notification, logging, access control relaxations/restriction, and more depending on application requirements. We have identified five properties which a CAAC model needs to satisfy:

1. *Responsiveness*: The system should immediately respond to any critical event.

2. *Correctness*: A change in policies should only be in response to a critical event.

3. *Non-Interference*: Policy changes due to critical events should not affect the normal working of the system.

4. *Liveness*: The duration of policy changes should be finite and last only as long as needed.

5. *Non-Repudiability*: All system activity should be non-repudiable.

The rest of the paper is organized as follows: Section 2 presents the preliminaries which describes the requirement for CAAC. Section 3 describes the notion of criticality and presents the CAAC architectural framework. In Section 4 we illustrate an example scenario of CAAC. Section 5 presents sample access control policies for CAAC followed by their validation in Section 6. In Sections 7, 8, and 9, we discuss various implementation specific issues of CAAC, related work and conclusions respectively.

## 2 Preliminaries

A CAAC model has greater requirements than traditional CA-RBAC access control, because of their ex-

[1]It is important to note, that this is a new framework for access control management which provides facilities for planning responses to critical events. It does not model the critical events themselves

plicit handling of critical events. Below we present some of the *functional requirements* of CAAC in detail which will provide a better understanding of the functions of the various components of our framework.

### 2.1 Role Management

The main functions of role management include: assigning proper roles to subjects and providing them with appropriate privileges based on their roles. Moreover, subject-to-role, role-to-role, and role-to-privilege constraints affect the subject-to-privilege mapping. Our architecture recognizes two basic types of roles: *System Roles* and *Space Roles*, similar to the model in [9]. The former is assigned to a subject when it becomes a part of the system, and is usually mapped based on the actual position held by the subject in the system, doctor in a hospital, for example. The latter however, is assigned to a subject based on the space it is currently inhabiting, like nurse in a hospital ED. A subject can have multiple space-roles, i.e. different roles in different spaces. The privileges are, however, only pertinent to the resources within the space the subject is inhabiting. It should be noted that for a subject, the system role is usually a subset of the space role. Hereafter, all references to access control are with respect to a smart space.

### 2.2 Context Management

The access control decisions made by a system have to reflect the changes that occur within the system. Context information makes it possible for the system to keep track of these changes. Context information can be roughly grouped into three categories: *subject context* (e.g.: location of the subject, subject's capability), *resource context* (e.g.: capability of the resource, current load on the resource) and *environmental context* (e.g.: number of subjects in a space at a given time). Access control models have to take all the aforementioned types of contexts into account when making access control decisions.

### 2.3 Critical Event Management

The effects of critical events, generated within a system, need real-time guarantees for alleviation and can not be generalized by generic context management. This is because critical events differ from other more mundane ones. Some of the principle differences are: 1) critical events require automatic change in access policies, unlike normal system state where contexts are evaluated on request, 2) all policy change with respect to critical events are temporary. However this is not a stringent requirement during non-critical contexts, 3) in critical contexts non-repudiability must be ensured.

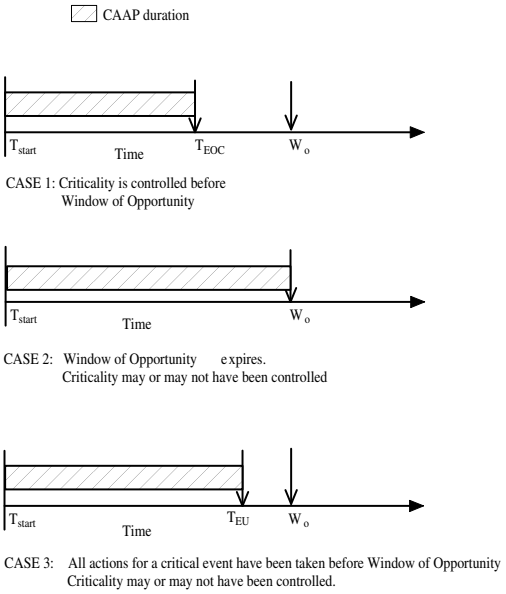


**Figure 1. Scenarios showing the required duration of CAAP mode**

## 3 Criticality

In order to be able to effectively model the automation of access control during critical events, we define a term called *criticality*. **Criticality** is a measure of the level of responsiveness in taking corrective actions to control the effects of a critical event and is used to determine the severity of critical events. To quantify this attribute, we introduce the term *Window-of-Opportunity* ($W_o$), which is an application dependent parameter defining the maximum delay that can possibly be allowed to take corrective action after the occurrence of a critical event. Therefore, lower the Window-of-Opportunity of a critical event the higher its criticality. A Window-of-Opportunity = 0 indicates maximum criticality for a critical event while a Window-of-Opportunity = $\infty$ indicates no criticality.

When a critical event occurs, the system is required to provide necessary facilities to take corrective actions against the critical event. The value of $W_o$, deter-

mined by the system for the critical event, specifies the time before which corrective actions need to be taken. Access policies implemented by the system, in its normal state, may not be able provide the necessary facilities to take corrective actions, against the critical event, within the required time duration ($W_o$). Therefore, it may be necessary for the system to implement a new set of access policies which facilitate timely action.

We refer to this new set of access policies as Criticality-Aware Access Policies (CAAP) and during their execution, the system is said to be in *CAAP-mode*. As the CAAP aids in handling the critical events in the system, the duration of CAAP-mode is bounded by $W_o$. However, if the critical event is controlled before $W_o$, the system should be removed from the CAAP-mode.

However, changing access policies, during critical events, may introduce security concerns. These stem from the fact that the new set of policies may provide higher privileges to access resources. If the system takes the extreme view of not changing the policies at all, then it may not be possible to control the critical event. If on the other hand the access policies are in the CAAP-mode for long periods of time, it may prompt misuse of the privileges. We therefore suggest the following criterion for managing the duration of the CAAP mode: 1) the window of opportunity ($W_o$), 2) the time instant when criticality is controlled ($T_{EOC}$), and 3) the time instant when all necessary actions to mitigate the criticality, have been taken ($T_{EU}$).

The rationale behind limiting CAAP-mode to $T_{EOC}$ stems from the fact that, once the critical event has been controlled, the system can return to its normal state of functioning where the CAAP is not required. Further, as the CAAP affect access to resources, once all the corrective actions have been taken, continual provision of CAAP is unnecessary irrespective of the outcome. Therefore, the maximum duration for which the system can be in the CAAP-mode, $T_{CAAP}$, is given by: $T_{CAAP} = min(W_o, T_{EU}, T_{EOC})$.

Figure 1 shows the three scenarios and the length of the required CAAP-mode. In CASE 1, the criticality has been controlled before the $W_o$ at time $T_{EOC}$ limiting the $T_{CAAP}$ to this time. In CASE 2, the $W_o$ expires, thereby removing the system from CAAP-mode. Finally in CASE 3, all the actions required to contain a critical event have been tried, therefore the system is removed from the CAAP-mode (irrespective of the state of the critical event). Before moving on to the

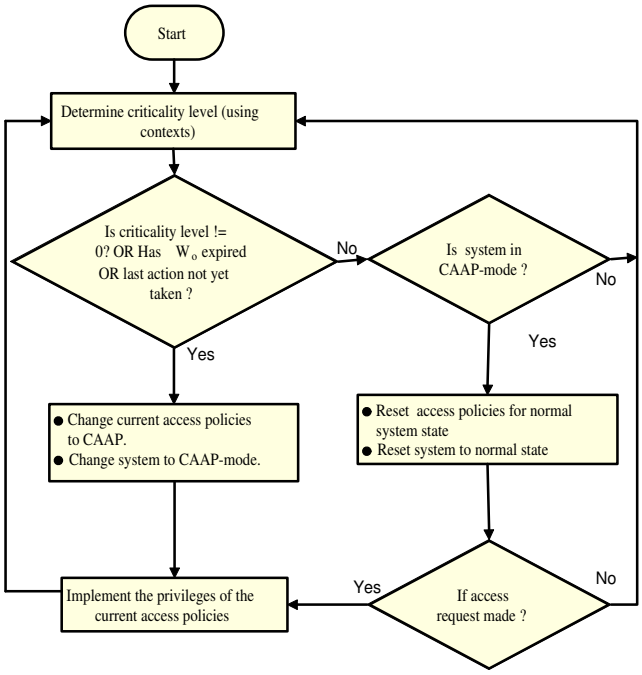


**Figure 2. Flow-chart for Criticality Handling in CAAC**

architectural framework of CAAC, we present a flow-chart (Figure 2) which shows the control flow of handling criticality in CAAC. The system continually determines the criticality level and on observing a critical event changes the access policies to CAAP and enters the CAAP-mode. If there is no criticality, then the system checks if the it is in CAAP-mode, if so, it returns the system to its normal state and enforces the appropriate policies when access is requested.

### 3.1 CAAC Framework

In this section, we present the CAAC framework that incorporates criticality awareness in a generic CA-RBAC model. Figure 3 shows the various components of the CAAC framework, which we define in more detail, below.

***Roles, Privileges and Subjects***: These components store information on the subjects in the system, the types of roles they are assigned to and the privileges assigned for each role.

***Access Control Meta-data***: This component abstracts the information from the previous three components and describes the dependencies between them. Example, in a hospital system a subject S1 (John Doe)

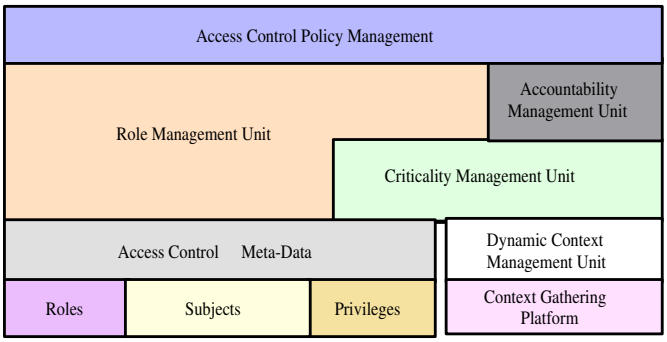


**Figure 3. CAAC Framework**

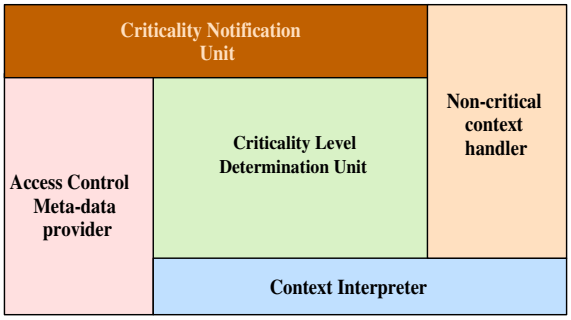


**Figure 4. Criticality Management Unit**

with a role R1(nurse) cannot be assigned a role R2 (doctor). In the literature these dependencies have also been called as constraints.

***Context Gathering Platform***: This is the component which collects raw context data from the system. This information is highly system/application dependent and belong to the subjects, resources and environment of the system.

***Dynamic Context Management Unit***: This component provides context management functions. It takes the raw context data, processes it and obtains a higher level context from it.

***Role Management Unit***: This component implements the role management functions for the system. It takes input from the access control meta-data and contextual information (both critical and non-critical) to dynamically enforce the privileges to subjects.

***Accountability Management Unit***: This is used to log all the system activity with respect to access control and is used for process improvement and accountability.

***Access Control Policy Management***: This is the highest component of the system and uses the underlying infrastructure to implement the access control and administrative policies for the system.

***Criticality Management Unit***: Given this overview of CAAC, we present a detailed description of the *Criticality Management Unit* which handles criticality of the system. Figure 4 shows its internal details

*Context Interpreter*: This is the component which monitors all the contextual information emanating from the system and intelligently detects the occurrence of a critical event.

*Criticality Level Determination:* Based on the input from the Context Interpreter, this component determines the level of criticality for an event. It also determines the maximum time ($W_o$) for which CAAP has to be enforced within the system.

*Criticality Notification Unit*: This component moves the system into CAAP-mode and informs other components about the associated access policy changes.

*Access Control Meta-data Provider*: This component is used to provide access control meta-data to the Criticality Notification Unit for determining the policies for the CAAP-mode.

*Non-Critical Context Handler* : This component is used for querying specific contextual information during normal system operation. In the event of any access request, this component is used as an interface to the Context Interpreter for obtaining the latest context information.

## 4 Example of Criticality Based Access Control

To understand CAAC better we present an example scenario for smart hospital emergency department (ED). Within this hospital system, each medical professional(MP) is assigned a *system-role*, based on their functions in the hospital (doctor, nurse, cardiologist). Each medical professionals is also assigned a *space-role* based on which area of the hospital they work in, doctors in the smart-ED get have roles like ED-surgeon and so on. This space role is usually generated based on the MP's system-role and other contextual information (time of entry into smart space). The space-role of a MP decides what privileges they have in accessing different resources in the smart-ED.

In our case, the space roles are generated based on the number of beds in the smart space. If there are N beds in the room, then the roles are $< ED - MP - Bed1 >, < ED - MP - Bed2 >, \ldots,$

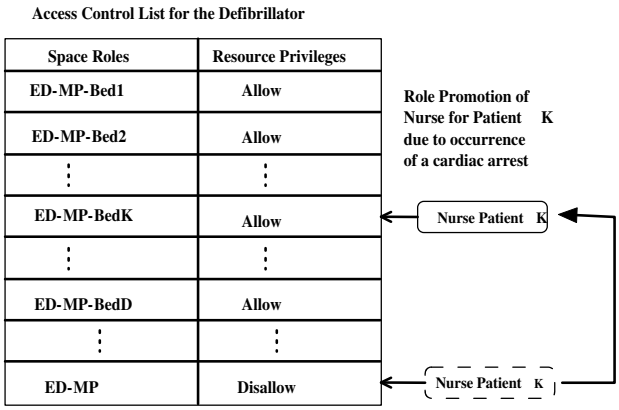


**Figure 5. Role Promotion due to Criticality**

$< ED - MP - BedN >$ and $< ED - MP >$. The first N space roles provides privileges which allow the usage of the device on the patient in the appropriate bed while the last role prohibits the device usage, but can allow access the common devices in the space. When a doctor or nurse enters our smart ED they get the space role of $< ED - MP >$,until they request access to a device pertaining to a patient in a particular bed.

When the smart-ED observes a critical event it varies its normal state access policies for managing it. To illustrate this fact we consider a sample critical event where a patient suddenly experiences cardiac arrest and her assigned MPs are not available.

If suppose patient3 experiences life-threatening ventricular fibrillation or ventricular tachycardia, the system will immediately, move to the CAAP-mode and implement the following new policies: issue a $code - blue$ warning, notify and promote the role of a medical professional who is in the vicinity of the patient, and is proficient in the necessary treatment to the space role $< ED - MP - Bed3 >$ (for the defibrillator), and computes the $W_o$ for the criticality based on the patient's state. A non-critical access policy would not have provided the MP with required privileges for accessing the defibrillator unless an explicit request was made which would have wasted valuable time and also proved distractive to the care providers.

The role of subject, for the defibrillator, is demoted (normal mode of operation) once the patient's health has been brought under control $T_{EOC}$. The subject role is also demoted at the time instant when all treatment for the patient has been tried and failed $T_{EU}$, in our case the medical professional has used the defibrillator at all three energy levels of 200,300 and 360 Joules and the patient's condition has not improved [1]. Figure 5 shows the role promotion in the defibrillator's access control list (ACL) (which maintains a record of roles and privileges). During this critical event management, the system maintains detailed logs listing all role promotions and associated activities. These logs will be used for maintaining accountability within the system.

## 5 Access Control Model and Policies

In this section, we present sample access control model and policies for CAAC framework. As mentioned before, when the system observes a critical event, it starts implementing a new set of access policies called CAAP that facilitates in tackling the situation. Here, we assume that on observing a critical event, the system assigns higher privileges to subjects as a part of CAAP by promoting the space-role. We begin with the access control model of our system, following which we delve into the policies themselves.

### 5.1 Model

The sample model presented here is based on the work done in [9] and extends it to include criticality. The system maintains an access control list (ACL) for each resource in its domain. An ACL for a resource, lists all possible space-roles that can be assigned to subjects using it and their corresponding set of access privileges (also called method). Each subject is granted privileges from the ACL corresponding to its space-role. For providing higher privileges to a subject (during criticality), the system simply promotes the subject's space-role to one with the required privileges rather than change the privileges associated with the subject's original space role. This makes the system easier to deploy, maintain and prevents erroneous modification of the privileges associated with the space-role of a subject.

Access to resources in our system is provided in two ways- *reactively*, on receiving an explicit request from a subject and *automatically*, on the occurrence of a critical event. In the case of reactive access, the system requires the subject to present a credential for justifying its request. As in [9], we rely on three types of credentials- the role of subjects, ownership of resources, executability of the method (corresponding to

the role) on a resource. However during automated access, the system automatically provides elevated privileges to subjects without the need for presenting any credentials. Given this access control model we now present a semi-formal specification for it.

### 5.2 Policy Specification

Access control model is specified in terms of policies. These policies define the rules for manipulating the ACLs to control the access to resources. In our system, policies are of two types:

*Access Control Policies* - these define the rules for controlling the access to different resources within the system. All the decisions take into account the roles, context and criticality for making the access control decisions.

*Administrative Policies* - these define the rules for system administration functions. It is used for adding subjects to the system, assigning them roles, mapping between the roles and maintaining accountability within the system.

#### 5.2.1 Access Control Policies

In this section we specify the access control policies of our system. Table 1 presents a list of symbols used in our specification.

**Table 1. Notations Table**

| Symbols | Description |
|---|---|
| $SysAdm$ | system administration role. |
| $R_{sys}$ | set of System roles. |
| $R_{space}$ | set of Space roles ($R_{sys} \subseteq R_{space}$). |
| $S$ | set of resources in the system. |
| $CONTEXT$ | set of contexts in the system. |
| $r_{sys}$ | system role ($r_{sys} \in R_{sys}$). |
| $r_{space}$ | space role ($r_{space} \in R_{space}$). |

Each time a subject enters a space, it gets a space-role which is generated from - the subject's system-role, and context information (subject,resource and environmental).

$currentrole(r_{sys}, context)$:$R_{sys} \times CONTEXT \rightarrow R_{space}$

This $currentrole$ function is derived from the one given in [9] by including contextual information in the system-role to space-role mapping. When a subject requests for a resource within a smart space, the system executes the Access Control Predicate to determine the level of access to be provided for the resource. The specification of this predicate is done in the guarded command language [2].

**Access Control Predicate**: In this predicate (Algorithm 1), when a subject $u$ makes a request to a resource $s$ for a specific method $m$ (thus making $request(u, s, m)$ true), it presents a set of credentials. The credentials are of following types: $typeof(u, r_{sys})$ which ensures that $u$ has the system-role $r_{sys}$, and $exports(s, m)$ which ensures that $s$ provides the access privileges $m$. Given these credentials, the predicate validates the access control by first computing the subject's space-role (using $currentrole$ function). It then uses the $promoterole$ function to decide if the role needs to be promoted. Role promotion happens only if the request is made in the $CAAP$ mode. The access is provided only if $m$ matches the space-role returned by $promoterole$ in the $ACL$ of $s$[2].

This predicate extends the access control predicate in [9], to include $promoterole$ function which handles criticality by executing the CAAP.

**Promote and Demote Role**: The function $promoterole$ (Algorithm 2) is used to promote the space-role of a subject with respect to a resource in case of an access request for mitigating a critical event. When invoked, this function checks the level of criticality using the function $Criticality$. If access is requested in a normal state ($Criticality() = 0$), it simply returns $r_{space}$. If a critical event has occurred, it does the following: 1) computes the window-of-opportunity of criticality using the function $calculatetime$ which takes the level of criticality as input, 2) implements CAAP, by computing the promoted space-role $r_{pSpace}$, based on the level of criticality using $calculaterole$, and 3) updates a Promoted Role Table (PRT) with the following tuple <*subject's id u, promoted role* $r_{pSpace}$*, start time of role-promotion* $currentTime()$*, stop time of role-promotion* $W_o$>. The PRT or Promoted Role Table is used to account for all subjects whose roles have been promoted. The presence of such a table allows for easy auditing and role accountability.

We also define another function called $demoterole$ (Algorithm 5) which demotes the space-role for

[2]The assumption here is that the privileges associated with a higher role encompasses all the lower privileges.

**Algorithm 1:** ACCESS CONTROL PREDICATE: Predicate is used for providing access control on explicit request from a subject (reactive)

$request(u, s, m) \wedge typeof(u, r_{sys}) \in C \wedge (s \in S) \wedge exports(s, m) \wedge$
$(promoterole(currentrole(r_{sys}, context), u), m) \in ACL_s \rightarrow true$

**Algorithm 2:** PROMOTE ROLE: Promotes the role of a subject depending upon level of criticality

**Function Name** : $promoterole$
**Attributes** : $u \in$ Set of Subjects, $r_{space} \in R_{space}$
**Return Value** : Promoted Role

**if** (Criticality() != 0) **then**
  $W_o = calculatetime(Criticality())$
  $r_{pSpace} = calculaterole(Criticality(), u)$ // Compute new space-role
  $PRT = PRT \cup \{(u, r_{pSpace}, currentTime(), W_o)\}$ // Update PRT
  return $r_{pSpace}$
**else**
  return $r_{space}$
**end if**

**Algorithm 3:** NOTIFICATION: Monitors the system for criticality, provide notification, issues orders for role promotion and maintains promotion logs

**Function Name**: $notifyCritical$

**while** (TRUE) **do**
  **while** (Criticality() = 0) **do**
    **if** ($state = CAAP\text{-}mode$) **then**
      $state = normal$ // Revert to the normal state, when criticality is over
      $demoterole(u)$
    **end if**
  **end while**
  **if** ($state = CAAP\text{-}mode$) **then**
    **if** (timer $W_o$ expired) **then**
      $state = normal$ // Revert to the normal state, when window of opportunity expires
      $currentrole(r_{sys}, context)$, for each resource
    **else**
      **if** (all actions taken) **then**
        $demoterole(u)$ // Revert to the normal state, when all actions have been taken
        $state = normal$
      **end if**
    **end if**
  **end if**
  $state = CAAP\text{-}mode$ // Criticality is observed, enter CAAP-mode
  $CL = Criticality()$
  $W_o = calculatetime(CL)$
  **if** !($notify(findUser(), CL, W_o)$) **then**
    $start\_timer(W_o)$
    $promoterole(currentrole(r_{sys}, context), u)$, for each resource
  **end if**
**end while**

**Algorithm 4:** ACCOUNTABILITY: Returns the current and promoted role of a subject

**Function Name** : $roleaccountability$
**Attributes** : $u \in$ Set of Subjects, $u_a \in$ Set of Subject

1: **if** $(typeof(u_a, SysAdm) \in C)$ **then**
2: $\forall a \in PRT, if\, u \in a$, obtain the tuple $(u, r_{space}, t_{start}, t_{end})$ // Access PRT
3: $\forall a \in CRT, if\, u \in a$, obtain the tuple $(u, r_{sys}, r_{space})$ // Access CRT
4: **end if**

**Algorithm 5:** DEMOTE ROLE: Demotes the role back to the original space role for all the resources

**Function Name** : $demoterole$
**Attributes** : $u \in$ Set of Subjects
**Return Value** : Demoted Role

1: **for** (each resource) **do**
2: $currentrole(r_{sys}, context)$, where $typeof(u, r_{sys}) \in C$ // Demote role
3: $\forall i \in PRT$, if $currentTime() < W_o$ such that $W_o \in i$ // Update PRT
4: $PRT = PRT - \{(u, r_{space}, t_{start}, W_o)\}$
5: $PRT = PRT \cup \{(u, r_{space}, t_{start}, currentTime())\}$
6: **end for**

a promoted subject when the critical event has been controlled. If the current time returned by $currentTime()$ is less than $W_o$ it updates the <stop time of the role-promotion> element of the corresponding PRT entry to the current time. This is done because PRT is used to record the duration when a subject had elevated access in the system only.

The system needs to continuously monitor for the occurrence of critical events in order to decide when to enter and exit the CAAP-mode. In our specification, this is achieved by the function $notifyCritical$.

**Notification**: The function $notifyCritical$ (Algorithm 3) has following five aspects:

1) To continuously monitor the system for critical events (using $Criticality$ which returns 0 when there is no criticality) and start the CAAP when a critical event occurs.

2) Identifying the appropriate subjects who can deal with a critical event using the function $findUser$.

3) Notifying the subjects identified to handle criticality. The $notify$ function is used for this purpose. It takes as input the result of $findUser$, the current criticality level and the associated $W_o$. If the notification has already been sent to the subject for the same criticality level, it returns false otherwise it returns true.

4) Promoting the roles of the subjects (using $promoterole$) to provide them necessary privileges, on resources, for handling the critical event.

5) Demoting the subject roles when the effects of critical events are handled or go beyond control, using the $demoterole$ function and returning the system to normal state.

Specifications for the functions $Criticality$, $findUser$, $notify$, $calculaterole$, $calculatetime$, and $start\_timer$ are application dependent and therefore are abstracted out.

### 5.2.2 Administrative Policies

For adding/removing subjects, roles and managing the smart spaces within the system, we use the policies given in [9]. We however incorporate an additional policy for maintaining accountability (Algorithm 4) within the system. The function basically, returns the details of the PRT and a Current-Role-Table (CRT table is used to store the current space role of a user) for a particular user. The presence of role accountability allows the administrator to find out which roles were promoted, when they were promoted and for what resources.

## 6 Validation

In this section we present informal proof sketch for validating the policy specifications given above. In all our proofs, we assume all access control policies execute correctly, all the administrative entities are trusted and the policies and system log cannot be accessed in a unauthorized manner. It needs to be emphasized that

space roles of subjects are not promoted in normal system state, but only during CAAP-mode.

**Theorem 6.1** *The system can enter CAAP-mode if and only if there is a critical event.*

**Proof** First, we prove the $if$ part. If there is a criticality, function $promoterole$ is called (in line 24 of Algorithm 3) and line 3 of $promoterole$ (Algorithm 2) will be executed.
Now we prove the $only - if$ part. If the role of a subject is promoted, it means that line 4 of $promoterole$ has been reached earlier and this can happen only in case of a critical event. As $promoterole$ implements CAAP, the result follows. ∎

The above theorem validates that if there is no criticality, the access control policies and the administrative policies are not affected due to the process of role-promotion thus satisfying the *Correctness* requirement of CAAC. The next theorem proves that any role promotion done on a subject's role can be promoted (when a critical event occurs) only for a finite amount of time, satisfying *Liveness* requirement of CAAC.

**Theorem 6.2** *For a single critical event, a subject's role is promoted for a maximum of $W_o$ time (i.e $max(T_{CAAP}) = W_o$), where $W_o$ is the window-of-opportunity to control the effects of the critical event.*

**Proof** From Theorem 6.1 and the assumption than roles are not promoted in normal system state, it follows that role can not be promoted in non-critical context i.e. when a subject's role is promoted, it is evident that a $W_o$ timer has been started for the critical event for which the role has been promoted. The promoted role of a subject is demoted in the following cases:
1) Line 11 of Algorithm 3: If the window-of-opportunity expires. Here $T_{CAAP} = W_o$.
2) Line 5 of Algorithm 3: If there is no criticality, but the system state is in CAAP-mode. This can only happen if a critical event has been controlled before its window-of-opportunity ($T_{EOC} < W_o$). Therefore, $T_{CAAP} = T_{EOC}$.
3) Line 14 of Algorithm 3: If the window-of-opportunity has not expired, but all actions for handling the critical event have been taken i.e. $T_{EU} < W_o$. Then $T_{CAAP} = T_{EU}$, where $T_{EU}$ is the instant when all the action have been taken.
Therefore for a single critical event, the subject's role is promoted for a maximum of $W_o$. ∎

As the role promotion is done for a finite amount of time, it should be notified immediately and not hinder the handling of the effects of a critical event, the following theorem proves this property satisfying the *Responsiveness* requirement of CAAC.

**Theorem 6.3** *When a critical event occurs - 1) the subject is immediately notified, 2) if required the subject's access privileges are elevated (role promotion), and 3) any role promotion is active until either the criticality is controlled or it cannot be controlled any more.*

**Proof** The proofs of the claims above are as follows:
1) When there is a criticality, the subjects are notified in line 22 of Algorithm 3.
2) If the subject being notified already has required privileges, its role is not promoted as the call for the function $calculaterole$ in line 3 of Algorithm 2 does not return elevated space-role (by definition). Otherwise, the function $calculaterole$ returns an elevated space-role based on the level of criticality, thus promoting the subject's role.
3) From Theorem 6.1 and the assumption that roles are not promoted during normal system state, we know that role promotion is done when there is a critical event and from Theorem 6.2 it follows that role is promoted until either the criticality is controlled or the time to take preventive action ($W_o$) expires. ∎

Until now, we have concentrated on the system functionality, during the occurrence of critical events. Now, we focus on the safety of the system. It follows from Theorem 6.1 that the safety of the system is not compromised if there is no critical event, satisfying the *Non-Interference* requirement of CAAC. However, when there is a critical event, the process of role-promotion can make the safety of the system vulnerable to potential malicious activities of the subject whose role is promoted. The following theorem explores the effects of role-promotion while handling criticality on the level of safety provided by the system. We prove that by providing *non-repudiation* capabilities and restricting the role promotion to a finite amount of time, any malicious intent is deterred and the system is safeguarded satisfying the *Non-Repudiation* requirement of CAAC.

**Theorem 6.4** *Malicious use of promoted role after the occurrence of a critical event is non-repudiable and limited to a finite amount of time.*

**Proof** Line 4 in Algorithm 2 and line 5 Algorithm 5 ensure that whenever a role is promoted it is recorded in PRT along with the appropriate start and end times enforcing non-repudiation of any malicious activity by a subject due to role promotion. As we assume that all the access control policies execute correctly, the PRT table is accurately updated. Further, as the PRT is assumed to be secured from any unauthorized access, and the administrator is a trusted entity, line 2 of Algorithm 4 can be used for ensuring non-repudiation.
From Theorem 6.1 and Theorem 6.2, it follows that in occurrence of a critical event, the maximum time the role can be promoted is $W_o$ thereby limiting potential malicious activity to a finite amount of time. ∎

## 7 Discussion

An access control system has three main aspects - context collection, access control decision making (which includes what decision to make and by whom) and enforcing the decisions. So far, in the previous section we have given details about what decisions are made in CAAC and how they are enforced. In this section we concentrate on the remaining aspects - context collection and deciding the access control decision making component.

One of the ways to gather contextual information is by using a network of wireless sensors. Recent advancement of MEMS technology has enabled sensors to be smart, tiny and communication enabled, thereby making them ideal for monitoring contextual information. The context information collected by the sensors can be aggregated, within the network, to produce composite contextual information, which enables the system to determine the presence and level of criticality. Our system requires continuous system monitoring for the occurrence of criticality. Individual sensors have very limited capabilities (low battery, processing, memory), therefore any continuous monitoring of context information may not be feasible [14]. The access control policies therefore may have to be tuned such that automatic criticality monitoring is optimized to increase the sensor network lifetime. Though any such optimization is application dependent, a standard parameter that can be used is the knowledge of the *frequency of criticality within the system*. Knowing this parameter, the system can monitor (collect contextual information) only during the time when it expects a criticality to occur. If the probability of occurrence of a critical event at a given time is $p$, then the frequency of monitoring is directly proportional to $p$. The usage of sensor network in our access control system also imposes certain other issues such as: network reliability, fault-tolerance, communication security . Though they do not affect our policies directly, however they will affect the actual working of the system in terms of context collection [14].

Access control decisions in a system can be made in two ways: centralized and distributed. Centralized schemes require the presence of a central controller entity which collects all contextual information and makes the requisite access control decisions based on the policies. Though they are easy to manage, they impose a single point of failure on the system. On the other hand, distributed schemes impose the decision making on each resource present within the system. Therefore each resource needs to be intelligent. This increases the complexity of individual resources, however one resource malfunction does not disable the system.

## 8 Related Work

Role Based Access control was first thoroughly studied in the seminal paper by Sandhu et al. [3]. This paper defined the basic components of RBAC such as user, roles, privileges, their interactions (constraints and hierarchy). By decoupling the process of directly associating privileges with a user, RBAC provided an effective and easy way of managing security within a system. Further it allowed easy implementation and enforcement of complex access control policies within the system. The concept of RBAC was generalized in [4] by incorporating subject roles, object roles and environment roles. As most systems are dynamic in nature, RBAC was further extended by including various context information in the access control decision making process. Some of the important work in CA-RBAC include [5] which considered the spatial, temporal and resource context in access control decision making, [7] presents team based access control model which is context-aware. The idea of context-sensitive access control was formally specified in [6], which attempted to perform access control based on the context of the requested operation. McDaniel [8] suggests that context specification in CA-RBAC is implementation dependent. Strembeck et. al. [11][12] provide an integrated framework to engineer and enforce context

constraints in RBAC.

Sampamane et. al. [9] present context-aware access control policies for smart spaces. They only consider spatial and subject context and define mode of access as *Individual*, *Shared* or *Collaborative* depending on the access privileges and the number of the subjects in the active space. The ideas from this work were further extended and implemented in [10], which presents an infrastructure for context aware access control and authentication in smart spaces. A dynamic context aware access control scheme for distributed health-care applications was presented in [13]. All these schemes are to large extent reactive in nature and do not consider the requirement of automated services when a critical event occurs.

## 9 Conclusion

In this paper we presented a new framework for specifying access control policies in smart spaces called Criticality-Aware Access Control (CAAC). The framework is developed by extending the existing CA-RBAC model by including a new concept of criticality. We also presented an architecture for CAAC which contains a criticality management component for handling critical contexts. We identify five basic requirements for our system to meet while handling criticality. They are: *Responsiveness*, *Correctness*, *Non-Interference*, *Liveness* and *Non-Repudiability*. Further, we presented a detailed description of a sample access control model and policies built under the CAAC framework and proved that it satisfied the aforementioned five basic requirements. In this paper we have taken a decidedly temporal view in modeling criticality, however work needs to be done to study other modeling techniques.

## References


[1] L.Cook. "Staying current on defibrillator safety". *In Journal of Nursing (33)11*, Nov 2003.

[2] E.Dijkstra. "Guarded Commands, non determinacy and formal derivation of programs". *In Communications of the ACM (18)8*, 1975.

[3] R.Sandhu, E.J.Coyne, H.L.Feinstein and C.E.Youman. "Role Based Access Control Models". *In IEEE Computer*, Feb, 1996.pp 38-47

[4] M.J.Moyer and M.Abamad. "Generalized Role Based Access Control". *In Proc. of 21st Int. Conf. Distributed Computing System*, 2001

[5] M.J.Covington, W.Long and S.Srinivasan. "Secure Context-Aware Applications Using Environmental Roles". *In Proc. of 6th ACM Symp. on Access Control Models Tech.*, 2001

[6] A.Kumar, N.Karnik and G.Chafle. "Context Sensitivity in Role-based Access Control". *In ACM SIGOPS Operating System Review 36(3)*, July, 2002

[7] C.K.Georgiadis, I.Mavridis, G.Pangalos and R.K.Thomas. "Flexible Team-Based Organizational Access Control using Contexts". *In Proc. of 6th ACM Symp. on Access Control Models Tech.*, 2001

[8] P.McDaniel. "On Context in Authorization Policy". *In Proc. of 8th ACM Symp. on Access Control Models Tech.*, 2003

[9] G.Sampemane, P.Naldurg and R.H.Campbell. "Access control for Active Spaces". *In Proc. of ACSAC*, 2002

[10] J.Al-Muhtadi, A.Ranganathan, R.H.Campbell and M.D.Mickunas. "Cerberus: A Context-Aware Security Scheme for Smart Spaces". *In Proc. IEEE Percom*, 2003

[11] G.Neumann and M.Strembeck. "An approach to engineer and enforce context constraints in an RBAC environment". *In Proc. of 8th ACM Symp. on Access Control Models Tech.*, 2003

[12] G.Neumann and M.Strembeck. "An integrated approach to engineer and enforce context constraints in RBAC environments". *In ACM TISSEC 7(3)*, 2004, pp 392-427

[13] J.Hu and A.C.Weaver. "Dynamic, Context-aware Security Infrastructure for Distributed Healthcare Applications". *In Proc. 1st Workshop on Pervasive Security, Privacy Trust*, 2004

[14] I.F.Akyildiz, W.Su, Y.Sankarasubramaniam and E.Cayirci. "A Survey on Sensor Networks". *In IEEE Communications Magazine 40(8)*, 2002, pp 102-114